

# Web Forms

HTML allows the development of web-forms. The elements of a web-form can be found in nearly all programming environments. E.G. Select Boxes, text boxes, radio buttons, check boxes. Although the names may differ slightly from other programming environments, the intent remains intact.

Unlike programming languages, HTML can not process the web form! This is normally done with a CGI or server side scripting program.

### **Web-form Controls:**

- text box
- select box
- radio buttons
- check boxes
- buttons

### **Questions:**

*What sort of attributes can be used with the controls?*

*Under what circumstances would you use a radio button, list box, text box or check box?*

### Example

The following is a snippet of code from a HTML page. The snippet only shows the form.

```
. . .  
<form action="handle_form.php" method="POST">  
<br />Enter your Name:  
<input type="text" name="p_name"  
      maxlength="20" size="20" />  
<br />Tick the box if you want a surprise:  
<input type="checkbox" name="p_number" />  
<br /><input type="submit" name="submit" value="Press" />  
</form>  
. . .
```

≈≈≈

#### handle\_form.php

The following is a section of code that shows how to 'handle' the submitted data.


```
. . .  
if(isset($_POST['p_name']))  
    echo '<p>Your name is ' . $_POST['p_name'] . '</p>';  
else  
    echo '<p>You do not have a name</p>';  
. . .
```

#### Questions:

*What type(s) of controls will the web form display?* 

*What happens when you click the submit button?* 

*What does POST mean?* 

*Can the method attribute of the form tag have a different value?* 

### Form Data

A lot of your PHP coding will be creating forms and validating the data. For this reason, it is imperative that you understand some simple validation techniques.

The previous page used the **isset** function to determine if the variable had been set. The PHP manual defines the **isset** function as:

---

**isset** -- Determine whether a variable is set

**Description**


`bool isset ( mixed var [, mixed var [, ...]])`

Returns **TRUE** if *var* exists; **FALSE** otherwise.

If a variable has been unset with [unset\(\)](#), it will no longer be set. **isset()** will return **FALSE** if testing a variable that has been set to **NULL**. Also note that a **NULL** byte ("\\0") is not equivalent to the PHP **NULL** constant.

---

**What does this mean?**

It means that the **isset** function should only be used to check for the EXISTENCE  of a variable, not to check if the variable contains any data.

**Question:**

*What does that mean for the `handle_form.php` code from the previous slide?*

# Other Validation Checks

PHP provides a number of useful functions to help validate your data. **empty()** is a very useful language construct.

---

**empty** -- Determine whether a variable is empty

### Description

bool **empty** ( mixed var)

**empty()** returns **FALSE** if *var* has a non-empty or non-zero value. In otherwords, "", 0, "0", **NULL**, **FALSE**, array(), var \$var;, and objects with empty properties, are all considered empty. **TRUE** is returned if *var* is empty.

---

### Questions:




*How does the behaviour of the two code snippets shown below differ?*

```

. . .
if(isset($_POST['p_name']))
    echo '<p>Your name is ' . $_POST['p_name'] . '</p>';
else
    echo '<p>You don\'t have a name</p>';
. . .

```

```

. 
if(!empty($_POST['p_name']))
 echo '<p>Your name is ' . $_POST['p_name'] . '</p>';
else
    echo '<p>You don\'t have a name</p>';
. . . 

```

*What is the exclamation mark (!) preceding empty for?*

*Why is there a slash (\) in don't?*



# Magic Quotes

PHP does some magic sometimes, like magic quotes. The magic can be switched on or off, however this is done via the php.ini file. If you are using PHP on the local machine, then you can configure the php.ini file yourself for all the magic you like.

**PHP Version 6 does not have magic quotes.**

However version 6 has not yet been released, therefore it is important that we learn about this crazy concept of magic quotes.

If you have shared web-hosting (or you are doing COMP306), then you won't have access to the php.ini file. Therefore, you should ALWAYS write your scripts so that they work regardless of the type of magic that is, or is not enabled.

### What are magic quotes?

The use of magic quotes is related to databases and SQL. Imagine that the handle\_form.php script inserts the p\_name and p\_number variables into a database table. Typically our insert query would look like this:

```
$query = "INSERT INTO someTable  
VALUES ('$p_name', $p_number) ";
```

The PHP pre-processor substitutes the variables of \$p\_name and \$p\_number for their actual values.

```
$query = "INSERT INTO someTable  
VALUES ('$p_name', $p_number)";
```

So if `$p_name` had a value of *Brent* and `$p_number` had a value of *2010*, then our query would equate to the following:

```
$query = "INSERT INTO someTable  
VALUES ('Brent', 2010)";
```

But what happens if someone had typed in *O'Malley*. What would our query look like then?

```
$query = "INSERT INTO someTable  
VALUES ('O'Malley', 2010)";
```

It doesn't matter what the programming language is, when you've got an extra quote floating around you're going to have all the wrong type of magic, i.e. syntax errors.

### **The Good Folks**

The good folks at PHP realized that this was going to cause problems so they introduced magic quotes. If magic quotes is enabled, then all quotes would be 'escaped' with a slash. Thus when the user typed in *O'Malley*, the `$p_name` variable would contain the value *O\Malley*. (Note the slash).

With Magic Quotes enabled our query would then look like this:

```
$query = "INSERT INTO someTable  
VALUES ('O\Malley', 2010)";
```

With Magic Quotes enabled, our insert query is okay, but our display code is crooked.

```
if(!empty($_POST['p_name']))  
    echo '<p>Your name is ' . $_POST['p_name'] . '</p>';  
else  
    ...
```

In the situation where we typed in *O'Malley* (and with Magic Quotes enabled), the output to the web-browser will be:

*Your name is O\'Malley.*


This is somewhat undesirable, however PHP offers a function that can strip the slashes for us. The function is called `stripslashes` and can be used like this:

```
echo '<p>Your name is ' . stripslashes($_POST['p_name']) .  
    '</p>';
```

*Your name is O'Malley.*

Of course in the previous case where Magic Quotes was not enabled, we could use the PHP function `addslashes` to manually escape the data before adding it into our database queries.

### **Questions:**

*Would you prefer Magic Quotes enabled or not enabled and why?* 

*How does PDO and prepared queries negate the issue?* 

# Portability

It is important to write your scripts so that they can work regardless of whether PHP is doing any behind the scenes 'magic'. Writing code in this fashion means that it is more portable to other web-servers running PHP.

You can check if *magic quotes* is enabled on the server by calling the boolean function, `get_magic_quotes_gpc`. The function returns **TRUE** (1) if magic quotes is enabled, or **FALSE** (0) if not enabled. The gpc stands for `get`, `post`, `cookie`; which is the order of priority when *register globals* is enabled.


**Register Globals** is another setting that is absent from PHP Version 6. It is not enabled on the server for this class.

By checking if Magic Quotes is enabled you can write your PHP code so that it can be easily ported to other web-servers running PHP.

## SUMMARY

`get_magic_quotes_gpc`      `addslashes`      `stripslashes`

## Questions:

*What is register globals? How does this affect your programming, and what are the security implications?* 



### Control Structures

PHP is a fully fledged scripting language, and as such control structures are built into the language. Control structures affect the flow of logic, and this is commonly done with **if / else** constructs.

#### Example


```
if($mood == 'Happy '){
    echo '<p>It is good to see you are happy.</p>';
}
else{
    echo "<p>I 'll see if I can make you more cheerful.</p>";
}
```

#### Questions:

*What does `if($mood == 'Happy')` mean?*

*What is the difference between `==` and `=`?*  

*Under what lexical group do the `=` and `==` fall under?*


*Can you see a syntactical difference in the 2nd echo statement?* 

≈≈≈

#### Switch Statements

**if / else** statements are great if you only need to look at two or three conditions. If you have more conditions than this, then you should consider using a switch statement which provides a much neater method of presenting the required logic.


### Example


```
switch($mood)   
{  
  case 'Happy':  
    echo '<p>It is good to see you are happy.</p>';  
    break;  
  
  case 'upset':  
  case 'sad':  
  case 'angry':  
    echo '<p>That is no good.  What\'s wrong?</p>';  
    break;  
  
  case 'so-so':  
    echo '<p>I hear you...</p>';  
    break;  
  
  default:  
    echo "<p>I don't know how to respond!</p>";  
    break;  
}
```

Presenting the programming logic of multiple if / else statements in a **switch** is much easier to read, maintain and understand. In the example above if `$mood` is *Happy*, then we'll display the text '*It is good to see you are happy*'.

If the value of `$mood` is *upset*, *sad* or *angry* then we'll display the text '*That is no good. What's wrong?*' and so on.

### Questions:

*Does PHP implement the switch construct differently from other languages?* 

*What is the **default**: keyword used for?* 


*Could you write this in the corresponding if / else structure?*

# LOOPS

Loops allow the repetitive task of performing the same action again and again. Generally there are 3 types of loops. There are **while** loops, which continually loop until a certain exit condition is met.

### Example:

```
$year = 2010;
while($year < 2015)
{
    echo '<p> ' . $year . '</p> ';
    $year++;
}
```



≈≈≈


There are **for** loops where you build the number of iterations into the loop parameter.

### Example:

```
for($i = 0; $i < 6; $i++)
{
    $year = 2010 + $i;
    echo '<p> ' . $year . '</p> ';
}
```

Both of these examples achieve the same thing, but using different types of loops.

### Questions:

*Under what circumstances would you use a while loop over a for-loop?* 

*What is a third loop type that I haven't mentioned yet?*

### Arrays

An array can be thought of as an enumerated data-type or variable that can only have a pre-defined set of possible values. An example might be an array for the *days\_of\_week* where the possible values would be *mon, tue, wed, thu, fri, sat, sun*. A value of *square* or *pie* obviously makes no sense in this array variable.


Another use for an array is where you need to group similar, coexisting variables together. One example could be for a class list. Rather than store each student id in a separate variable, *\$student1, \$student2, ...*, we could store the entire class in ONE *\$student* array.

Although there may not be much difference in the syntax between using *\$student1* or *\$student2* as opposed to *\$student[1]* and *\$student[2]*, there is a massive difference in functionality.

#### **Questions:**

*What are the square brackets ed with \$student[1]? *

*What are the differences in functionality?  *

*Is there an example of when you wouldn't use an array? *

*What advantages do you have by using an array?*

# Array Indexes

Arrays always have an index and a value. In PHP the index is commonly referred to as the key. The key / index usually identifies the order of the elements inside the array. Array indexes usually start at 0, and this is the case with PHP.

### Example:

```
$myArray = array('Great', 'Happy', 'So-So', 'Sad');
```

In the `$myArray` example, the first element *'Great'* has an index of 0, *'Happy'* has an index of 1 and so on. Therefore to display *Great* and *Happy* to the web-browser, we would use code like this:

```
echo $myArray[0];           //display Great
echo $myArray[1];           //display Happy
```

PHP allows us to specify the starting index of an array as well, and we'll demonstrate that in the example below.

```
$myMonths = array(1 => 'January', 'February', 'March');
```

*January* has an index of 1, *February* 2 and so on. It is also possible to specify individual keys for each value, and those keys don't have to be numeric either.

```
$myArray = array('S0001234' => 'Smith',
                 'S0001235' => 'Jones');
```

# Array Functionality

Arrays allow for powerful programming code to be written that wouldn't normally be available for normal variables. For example if a programming language supports an array, then it will also provide a method to determine the number of elements there are in the array. PHP has two such methods of determining the number of elements in the array. **count** and **sizeof**.

With the combination of a count method for the array, and loops we can now start being very productive with arrays and PHP.

### Example:

```
$myArray = array('Great ', 'Happy ', 'So-So ', 'Sad ');  
  
for($i = 0; $i < (sizeof($myArray)); ++$i)  
{  
    echo '<p>The value $myArray[' . $i . '] is:&nbsp;';  
    echo $myArray[$i] . '</p>';  
}
```

### Questions:

*What is displayed to the browser?*

*Notice how I used ++\$i instead of the more familiar \$i++. What is the difference?*

*Some languages provide access to arrays via () style brackets, whereas others use [] style brackets. What is the better technique?*

### Array Functions

The previous example only worked because the array keys were numeric, and started at 0 with increments of 1. However, in the case of `$myArray` storing the student id as the key, the student's last name as the value, accessing the arrays as we did previously will not work.

#### Example:

```
$myArray = array('S0001234' => 'Smith ',
                 'S0001235' => 'Jones ');
```

`$i` is an integer 0, 1, 2 etc from the **for** loop, and `$myArray` is using a string ('S0001234' etc) as the index.

*//ERROR: This will not work!*

```
for($i = 0; $i < sizeof($myArray); $i++)
{
    echo '<p>The student id of ' . $i . '&nbsp;';
    echo 'belongs to ' . $myArray[$i] . '</p>';
}
```

≈≈≈

PHP provides a special **foreach** loop that allows you to access the contents of an array where the index is not a numeric value.

#### Example:

```
foreach($myArray as $key => $value)
    echo "<p>The student id of $key belongs to $value</p>";
```

#### Question:

*Do you understand how the **foreach** loop works?*


### Ranges

PHP also provides a special type of array, a range. A range can be used where you need to fill an array with a predefined set of numeric values.

#### Previous Examples:

```
$year = 2010;
while($year < 2015)
{
    echo '<p> ' . $year . '</p> ';
    $year++;
}
```

≈

```
for($i = 0; $i < 6; $i++)
{
    
    $year = 2005 + $i;
    echo '<p> ' . $year . '</p> ';
}
```

#### Range Example:



```
$year = range(2010, 2015);

for($i = 0; $i < count($year); $i++)
    echo '<p> ' . $year[$i] . '</p> ';
```

PHP allows you to use the **range** function with integers or characters, however if you use characters you can only specify characters of ONE character length. There is also an optional increment parameter. When left blank, PHP will default the increment to 1, however you can specify other positive increment values if you want.